Carga del dataset y definición del objetivo de regresión

En esta primera sección se realiza la **carga del conjunto de datos** y la definición de la **variable objetivo** (`TARGET`) sobre la que se entrenará el modelo de regresión.

Pasos principales:

1. **Importación de librerías:**
   Se utilizan `pandas` y `numpy` para la manipulación de datos, y `warnings` para suprimir mensajes no relevantes.
2. **Configuración inicial:**
   Se define una semilla aleatoria (`RANDOM_STATE = 42`) que asegura la reproducibilidad de los resultados.
3. **Carga del archivo CSV:**
   El dataset `dataset_diabetes.csv` se lee mediante `pandas.read_csv()` .
   - Se verifica que el archivo exista antes de su lectura.
   - Se imprime la estructura general del DataFrame (`df.info()`), mostrando el número de columnas y tipos de datos.
4. **Definición del objetivo de regresión:**
   La variable `BMI` (Índice de Masa Corporal) se selecciona como variable objetivo (`TARGET`), que será el valor que los modelos intentarán predecir.
5. **Tratamiento opcional de etiquetas:**
   Si el dataset contiene la columna `Diabetes_012` , esta se convierte en texto legible (`NO DIAB` , `PREDIAB` , `DIAB`) para facilitar la interpretación posterior.
6. **Separación de variables:**
   - `X` : contiene las variables predictoras (independientes).
   - `y` : contiene la variable objetivo (`BMI`).
7. **Resumen estadístico del objetivo:**
   Se muestra el tamaño del dataset, la media, desviación estándar y rango de valores del objetivo.

Esta preparación inicial permite verificar la integridad y estructura de los datos antes de aplicar las técnicas de regresión en las siguientes etapas.

In [2]:
```
# ========================================
# 1) Cargar datos y objetivo
# ========================================
import os, warnings
import numpy as np
import pandas as pd
warnings.filterwarnings("ignore")

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
```

```
DATA_FILE = "dataset_diabetes.csv"   # <-- tu archivo
TARGET    = "BMI"                                # objetivo continuo (regresión)
assert os.path.exists(DATA_FILE), f"No se encuentra {DATA_FILE}"

df = pd.read_csv(DATA_FILE)
print(df.info())

# (Opcional) Hacer legible la columna Diabetes_012 si existe
if "Diabetes_012" in df.columns:
    df["Diabetes_012"] = df["Diabetes_012"].map({0:"NO", 1:"PREDIAB", 2:"DIAB"})

y  = df[TARGET]
X  = df.drop(columns=[TARGET])

print(
    "Shape:", X.shape,
    "| y(mean):", round(y.mean(), 4),
    "| y(std):", round(y.std(), 4),
    "| y[min,max]:", (round(y.min(), 4), round(y.max(), 4))
)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_012          253680 non-null  object 
 1   HighBP                253680 non-null  int64  
 2   HighChol              253680 non-null  int64  
 3   CholCheck             253680 non-null  int64  
 4   BMI                   253680 non-null  int64  
 5   Smoker                253680 non-null  int64  
 6   Stroke                253680 non-null  int64  
 7   HeartDiseaseorAttack  253680 non-null  int64  
 8   PhysActivity          253680 non-null  int64  
 9   Fruits                253680 non-null  int64  
 10  Veggies               253680 non-null  int64  
 11  HvyAlcoholConsump     253680 non-null  int64  
 12  AnyHealthcare         253680 non-null  int64  
 13  NoDocbcCost           253680 non-null  int64  
 14  GenHlth               253680 non-null  int64  
 15  MentHlth              253680 non-null  int64  
 16  PhysHlth              253680 non-null  int64  
 17  DiffWalk              253680 non-null  int64  
 18  Sex                   253680 non-null  object 
 19  Age                   253677 non-null  float64
 20  Education             253680 non-null  int64  
 21  Income                253680 non-null  int64  
dtypes: float64(1), int64(19), object(2)
memory usage: 42.6+ MB
None
Shape: (253680, 21) | y(mean): 28.3824 | y(std): 6.6087 | y[min,max]: (12, 98)
```

In [8]:
```
X , y
```

Out[8]: (

```
        Diabetes_012  HighBP  HighChol  CholCheck  Smoker  Stroke  \
0                nan       1         1          1       1       0
1                nan       0         0          0       1       0
2                nan       1         1          1       0       0
3                nan       1         0          1       0       0
4                nan       1         1          1       0       0
...              ...     ...       ...        ...     ...     ...
253675           nan       1         1          1       0       0
253676           nan       1         1          1       0       0
253677           nan       0         0          1       0       0
253678           nan       1         0          1       0       0
253679           nan       1         1          1       0       0

        HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  ...  \
0                          0             0       0        1  ...
1                          0             1       0        0  ...
2                          0             0       1        0  ...
3                          0             1       1        1  ...
4                          0             1       1        1  ...
...                      ...           ...     ...      ...  ...
253675                     0             0       1        1  ...
253676                     0             0       0        0  ...
253677                     0             1       1        0  ...
253678                     0             0       1        1  ...
253679                     1             1       1        0  ...

        AnyHealthcare  NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  \
0                   1            0        5        18        15         1
1                   0            1        3         0         0         0
2                   1            1        5        30        30         1
3                   1            0        2         0         0         0
4                   1            0        2         3         0         0
...               ...          ...      ...       ...       ...       ...
253675              1            0        3         0         5         0
253676              1            0        4         0         0         1
253677              1            0        1         0         0         0
253678              1            0        3         0         0         0
253679              1            0        2         0         0         0

           Sex   Age  Education  Income
0       Female   9.0          4       3
1       Female   7.0          6       1
2       Female   9.0          4       8
3       Female  11.0          3       6
4       Female  11.0          5       4
...        ...   ...        ...     ...
253675    Male   5.0          6       7
253676  Female  11.0          2       4
253677  Female   2.0          5       2
253678    Male   7.0          5       1
253679  Female   9.0          6       2

[253680 rows x 21 columns],
0         40
1         25
2         28
3         27
4         24
          ..
253675    45
```

```
253676    18
253677    28
253678    23
253679    25
Name: BMI, Length: 253680, dtype: int64)
```

División del conjunto de datos (80% Entrenamiento / 20% Prueba)

En este paso se divide el dataset original en dos subconjuntos:

- **Conjunto de entrenamiento (`X_train` , `y_train`)**: utilizado para ajustar los modelos.
- **Conjunto de prueba (`X_test` , `y_test`)**: empleado posteriormente para evaluar el desempeño del modelo con datos no vistos.

La función `train_test_split()` del módulo `sklearn.model_selection` realiza esta división con una proporción del **80% para entrenamiento** y **20% para prueba**. Además, se utiliza una semilla (`RANDOM_STATE = 42`) para garantizar que los resultados sean reproducibles.

Finalmente, se imprimen las dimensiones de cada subconjunto para confirmar la correcta separación de los datos.

In [4]:
```
# ========================================
# 2) Split temprano (80/20)
# ========================================
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=RANDOM_STATE
)
print(f"Train: {X_train.shape} | Test: {X_test.shape}")
```

```
Train: (202944, 21) | Test: (50736, 21)
```

Preprocesamiento de variables (Imputación, Escalado y Codificación)

En esta sección se construye el **pipeline de preprocesamiento** que transforma los datos de manera uniforme antes de entrenar los modelos de regresión.

Componentes principales:

1. **Selección de variables:**
   - `cat_features` : columnas categóricas (tipo `object` , `category` o `string`).
   - `num_features` : columnas numéricas o booleanas.
2. **Imputación de valores faltantes:**
   - Numéricas → reemplazo por la **mediana**.

- Categóricas → reemplazo por el **valor más frecuente**.

3. **Estandarización y codificación:**
   - Numéricas → `StandardScaler` para normalizar escalas.
   - Categóricas → `OneHotEncoder` para convertir categorías en variables binarias.
4. **ColumnTransformer:**
   - Combina ambos procesos (numérico y categórico) en una sola estructura coherente.
5. **Pipeline general (`build_pipe`):**
   - Une el preprocesamiento con el modelo.
   - Incluye `VarianceThreshold(0.0)` para eliminar columnas sin variabilidad.
   - No se aplica **SMOTE**, ya que este flujo trabaja con regresión (no clasificación).

Finalmente, se imprimen las cantidades de variables numéricas y categóricas detectadas para asegurar que el preprocesamiento se configuró correctamente.

In [5]:
```
# ========================================
# 3) Preprocesamiento (en pipeline)
# ========================================
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_selection import VarianceThreshold
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as ImbPipeline

cat_features = X_train.select_dtypes(include=["object","category"]).columns.toli
num_features = X_train.select_dtypes(include=["number","bool"]).columns.tolist()

# OneHotEncoder compatible (con fallback de parámetro 'sparse')
try:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
except TypeError:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

num_pipe = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler",  StandardScaler()),
])

cat_pipe = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe",     ohe),
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", num_pipe, num_features),
        ("cat", cat_pipe, cat_features),
    ],
    remainder="drop",
)

def build_pipe(model):
```

```
    # En regresión NO se usa SMOTE
    return ImbPipeline([
        ("prep", preprocessor),
        ("var0", VarianceThreshold(0.0)),  # limpia columnas constantes tras OHE
        ("model", model),
    ])

print(f"Features numéricas: {len(num_features)} | categóricas: {len(cat_features
```

```
Features numéricas: 19 | categóricas: 2
```

Preprocesamiento de variables (Imputación, Escalado y Codificación)

En esta sección se construye el **pipeline de preprocesamiento** que transforma los datos de manera uniforme antes de entrenar los modelos de regresión.

Componentes principales:

1. **Selección de variables:**
   - `cat_features`: columnas categóricas (tipo `object`, `category` o `string`).
   - `num_features`: columnas numéricas o booleanas.
2. **Imputación de valores faltantes:**
   - Numéricas → reemplazo por la **mediana**.
   - Categóricas → reemplazo por el **valor más frecuente**.
3. **Estandarización y codificación:**
   - Numéricas → `StandardScaler` para normalizar escalas.
   - Categóricas → `OneHotEncoder` para convertir categorías en variables binarias.
4. **ColumnTransformer:**
   - Combina ambos procesos (numérico y categórico) en una sola estructura coherente.
5. **Pipeline general (`build_pipe`):**
   - Une el preprocesamiento con el modelo.
   - Incluye `VarianceThreshold(0.0)` para eliminar columnas sin variabilidad.
   - No se aplica **SMOTE**, ya que este flujo trabaja con regresión (no clasificación).

Finalmente, se imprimen las cantidades de variables numéricas y categóricas detectadas para asegurar que el preprocesamiento se configuró correctamente.

In [6]:
```
# ========================================
# 4) Modelos candidatos (REGRESIÓN)
# ========================================
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
```

```
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor

candidates = [
    ("LR",  LinearRegression()),
    ("RG",  Ridge(random_state=RANDOM_STATE)),
    ("LS",  Lasso(random_state=RANDOM_STATE, max_iter=5000)),
    ("EN",  ElasticNet(random_state=RANDOM_STATE, max_iter=5000)),
    ("KNR", KNeighborsRegressor()),
    ("DTR", DecisionTreeRegressor(random_state=RANDOM_STATE)),
    ("RFR", RandomForestRegressor(
        n_estimators=300, random_state=RANDOM_STATE, n_jobs=-1
    )),
    ("MLP", MLPRegressor(
        hidden_layer_sizes=(64,), max_iter=800, random_state=RANDOM_STATE
    )),
    ("XGB", XGBRegressor(
        tree_method="hist", random_state=RANDOM_STATE,
        n_estimators=400, learning_rate=0.05, max_depth=6,
        subsample=0.9, colsample_bytree=0.9, n_jobs=-1
    )),
    ("LGB", LGBMRegressor(
        n_estimators=500, learning_rate=0.05, max_depth=-1,
        subsample=0.9, colsample_bytree=0.9,
        random_state=RANDOM_STATE, n_jobs=-1, verbosity=-1
    )),
    ("CAT", CatBoostRegressor(
        iterations=600, learning_rate=0.05, depth=6,
        random_state=RANDOM_STATE, l2_leaf_reg=3.0,
        verbose=False, allow_writing_files=False, thread_count=-1
    )),
]
```

Entrenamiento inicial (Baseline) con Validación Cruzada

En este bloque se realiza la **evaluación inicial de los modelos candidatos** mediante **validación cruzada (Cross-Validation)**, sin ajustar aún los hiperparámetros.

Detalles del proceso:

1. **Validación cruzada K-Fold:**
   Se divide el conjunto de entrenamiento en 5 particiones (folds) utilizando `KFold` para estimar la capacidad de generalización de cada modelo.

2. **Métricas de evaluación:**

   - **RMSE:** raíz del error cuadrático medio (*Root Mean Squared Error*).
   - **MAE:** error absoluto medio (*Mean Absolute Error*).
   - **R^2:** coeficiente de determinación (qué tan bien el modelo explica la variabilidad del objetivo).

3. **Cross-validation automática:**
   Con `cross_validate()`, se entrena y evalúa cada modelo candidato, promediando los resultados en las 5 divisiones.

4. **Comparación de desempeño:**
   Se guardan los resultados en un DataFrame (`baseline_df`) ordenado por RMSE y se muestra el **modelo con mejor desempeño promedio** como "Baseline ganador".

Este paso permite identificar qué modelo ofrece mejor rendimiento inicial antes de pasar a una fase de ajuste de hiperparámetros o análisis más profundo.

In [13]:
```
# ========================================
# 5) Baseline con CV (sin tuning)
# ========================================
from sklearn.model_selection import KFold, cross_validate
import numpy as np
import pandas as pd

cv = KFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
scoring = {
    "rmse": "neg_root_mean_squared_error",
    "mae":  "neg_mean_absolute_error",
    "r2":   "r2",
}

rows = []
for name, model in candidates:
    pipe = build_pipe(model)
    scores = cross_validate(
        pipe, X_train, y_train,
        cv=cv, scoring=scoring,
        n_jobs=1, pre_dispatch=1,   # adaptación: evita SIGKILL por memoria
        error_score=np.nan
    )
    row = {
        "model": name,
        "rmse": -scores["test_rmse"].mean(),
        "mae":  -scores["test_mae"].mean(),
        "r2":    scores["test_r2"].mean(),
    }
    rows.append(row)
    print(f"{name:>3} | RMSE {row['rmse']:.3f} | MAE {row['mae']:.3f} | R² {row[

baseline_df = pd.DataFrame(rows).sort_values("rmse")
try:
    display(baseline_df)
except Exception:
    print(baseline_df)

baseline_best_name  = baseline_df.iloc[0]["model"]
baseline_best_model = dict(candidates)[baseline_best_name]
print(f">>> Baseline ganador: {baseline_best_name}")
```

```
 LR | RMSE 6.195 | MAE 4.373 | R² 0.122
 RG | RMSE 6.195 | MAE 4.373 | R² 0.122
 LS | RMSE 6.464 | MAE 4.600 | R² 0.044
 EN | RMSE 6.382 | MAE 4.526 | R² 0.068
KNR | RMSE 6.653 | MAE 4.734 | R² -0.013
DTR | RMSE 8.642 | MAE 5.962 | R² -0.709
RFR | RMSE 6.498 | MAE 4.616 | R² 0.034
MLP | RMSE 6.064 | MAE 4.249 | R² 0.159
XGB | RMSE 6.045 | MAE 4.236 | R² 0.164
LGB | RMSE 6.042 | MAE 4.232 | R² 0.165
CAT | RMSE 6.035 | MAE 4.227 | R² 0.167
```

	model	rmse	mae	r2
10	CAT	6.034799	4.227124	0.166982
9	LGB	6.042153	4.232139	0.164946
8	XGB	6.045343	4.236197	0.164065
7	MLP	6.064009	4.249057	0.158901
1	RG	6.194997	4.373158	0.122166
0	LR	6.194997	4.373159	0.122166
3	EN	6.381747	4.526248	0.068455
2	LS	6.463702	4.600000	0.044379
6	RFR	6.497861	4.615527	0.034178
4	KNR	6.653366	4.733880	-0.012571
5	DTR	8.642370	5.961766	-0.708572

```
>>> Baseline ganador: CAT
```

Tuning de hiperparámetros con Validación Cruzada (selección rápida del ganador)

En esta sección se optimizan los **hiperparámetros** de varios modelos de **regresión** mediante **RandomizedSearchCV** y **validación cruzada (K-Fold)**. El objetivo es encontrar, para cada algoritmo, la combinación de parámetros que **minimice el RMSE** y luego **elegir el modelo ganador**.

Estrategia

- **Pipelines**: cada búsqueda se ejecuta sobre `build_pipe(model)`, que incluye el **preprocesamiento** (imputación, OHE, escalado, eliminación de varianza nula) + el **modelo**.
- **Esquemas de CV**:
  - `cv_light` (5-fold) para modelos livianos (Ridge/ElasticNet).
  - `cv_heavy` (3-fold) para modelos más costosos (Random Forest, XGB, LGB, CAT) para **reducir tiempo y memoria**.
- **Espacios de hiperparámetros** (`param_spaces`) específicos por modelo: alfas y *l1_ratio* (modelos lineales), número de estimadores, profundidad, *learning_rate*, etc.

- **Métricas**: se evalúan `rmse`, `mae` y `r2`, y se define `refit="rmse"` para que el mejor estimador quede ajustado según **RMSE**.
- **Ejecución controlada**: se fijan `random_state` para **reproducibilidad** y `n_jobs/pre_dispatch` para **cuidar memoria**.

Flujo

1. Para cada modelo en `to_tune`, se lanza un **RandomizedSearchCV** con `n_iter` acotado (más iteraciones si es "heavy").
2. Se **entrena** sobre `X_train, y_train` y se **recupera** el mejor pipeline y sus hiperparámetros.
3. Se convierte el puntaje de `neg_root_mean_squared_error` a **RMSE positivo** y se **ordena** la lista por menor RMSE.
4. Se **imprime** el **ganador optimizado** con su `RMSE (CV)` y los **mejores hiperparámetros**.

> Resultado: un **modelo ganador** (y su pipeline) listo para reentrenar en *train* y evaluar posteriormente en *test*.

In [9]:
```
# ========================================
# 6) Tuning con CV y elección del ganador (rápido)
# ========================================
import tempfile, shutil
from sklearn.model_selection import RandomizedSearchCV, KFold
from scipy.stats import randint, uniform
try:
    from scipy.stats import loguniform
except Exception:
    from sklearn.utils.fixes import loguniform

cv_light = KFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
cv_heavy = KFold(n_splits=3, shuffle=True, random_state=RANDOM_STATE)

param_spaces = {
    "RG":  {"model__alpha": loguniform(1e-3, 1e3)},
    "EN":  {"model__alpha": loguniform(1e-3, 1e2), "model__l1_ratio": uniform(0.
    "RFR": {"model__n_estimators": randint(200, 600), "model__max_depth": randin
    "XGB": {"model__n_estimators": randint(250, 600), "model__learning_rate": lo
    "LGB": {"model__n_estimators": randint(300, 800), "model__learning_rate": lo
    "CAT": {"model__iterations": randint(300, 700), "model__learning_rate": logu
}

to_tune = [
    ("RG",  Ridge(random_state=RANDOM_STATE)),
    ("EN",  ElasticNet(random_state=RANDOM_STATE, max_iter=5000)),
    ("RFR", RandomForestRegressor(random_state=RANDOM_STATE, n_jobs=1)),
    ("XGB", XGBRegressor(tree_method="hist", random_state=RANDOM_STATE, n_jobs=1
    ("LGB", LGBMRegressor(random_state=RANDOM_STATE, n_jobs=1, verbosity=-1)),
    ("CAT", CatBoostRegressor(random_state=RANDOM_STATE, verbose=False,
                              allow_writing_files=False, thread_count=1)),
]

scoring = {"rmse": "neg_root_mean_squared_error", "mae": "neg_mean_absolute_erro
```

```
best_models = []
cache_dir = tempfile.mkdtemp(prefix="skcache_")
try:
    for name, base_model in to_tune:
        pipe = build_pipe(base_model)
        heavy = name in ["RFR","XGB","LGB","CAT"]
        search = RandomizedSearchCV(
            pipe, param_spaces[name],
            n_iter=(15 if heavy else 12),
            cv=(cv_heavy if heavy else cv_light),
            scoring=scoring, refit="rmse",
            n_jobs=1, pre_dispatch=1,  # cuida memoria
            random_state=RANDOM_STATE, verbose=1,
            error_score=np.nan, return_train_score=False
        )
        search.fit(X_train, y_train)
        best_models.append((name, search.best_estimator_, -search.best_score_, s
    best_models.sort(key=lambda x: x[2])  # menor RMSE primero
    best_name, final_pipe_opt, best_cv_rmse, best_params = best_models[0]
    print(f">>> GANADOR OPTIMIZADO: {best_name} (RMSE CV={best_cv_rmse:.3f})")
    print("Mejores hiperparámetros:", best_params)
finally:
    shutil.rmtree(cache_dir, ignore_errors=True)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
>>> GANADOR OPTIMIZADO: CAT (RMSE CV=6.037)
Mejores hiperparámetros: {'model__iterations': 613, 'model__learning_rate': 0.034
646653174710614}
```

Comparación entre modelo baseline y modelo optimizado (validación cruzada)

En esta sección se realiza una **comparación justa** entre el modelo base (baseline) y el modelo optimizado durante la etapa de *tuning*, utilizando el **mismo esquema de validación cruzada (K-Fold)**.

Objetivo

Determinar si el modelo ajustado mejora de forma **significativa** el desempeño frente al baseline, evitando sobreajuste o incrementos de complejidad innecesarios.

Detalles del proceso:

1. Se define un esquema `same_cv` con **5 particiones** (`KFold`), asegurando la misma semilla para ambos modelos.
2. Se evalúan ambos modelos (baseline y tuned) utilizando **RMSE** como métrica principal.
3. Se aplica una **regla de selección simple:**

- Si la mejora del modelo optimizado sobre el baseline es menor al **1%**, se mantiene el baseline (por simplicidad).
- Si la mejora es ≥ 1%, se elige el modelo ajustado como ganador.

El resultado final muestra el **modelo seleccionado para la evaluación en TEST**, junto con sus puntajes promedio de RMSE.

In [15]:
```
# ========================================
# 7) Comparación justa (solo CV) - baseline vs ganador
# ========================================
from sklearn.model_selection import KFold, cross_validate

same_cv = KFold(n_splits=5, shuffle=True, random_state=123)
pipe_baseline_best = build_pipe(baseline_best_model)
pipe_tuned_best    = final_pipe_opt

def cv_rmse(pipe, name):
    s = cross_validate(
        pipe, X_train, y_train, cv=same_cv,
        scoring={"rmse":"neg_root_mean_squared_error"},
        n_jobs=-1
    )
    rmse = -s["test_rmse"].mean()
    print(f"{name}: RMSE {rmse:.4f}")
    return rmse

rmse_base = cv_rmse(pipe_baseline_best, f"Baseline({baseline_best_name})")
rmse_tune = cv_rmse(pipe_tuned_best,   f"Tuned({best_name})")

# Regla: si la mejora < 1% del RMSE base, nos quedamos con el baseline (más simp
if (rmse_base - rmse_tune) / rmse_base >= 0.01:
    winner_name, winner_pipe = best_name, pipe_tuned_best
else:
    winner_name, winner_pipe = baseline_best_name, pipe_baseline_best

print(f">>> Modelo seleccionado para TEST: {winner_name}")
```

```
Baseline(CAT): RMSE 6.0362
Tuned(CAT): RMSE 6.0346
>>> Modelo seleccionado para TEST: CAT
```

Política de decisión y postprocesamiento de predicciones

En esta parte se define una **política de postprocesamiento** que controla cómo se tratan las predicciones generadas por los modelos antes de su evaluación final.

Elementos del diccionario `POLICY`:

- `"clip_to_train_range": True` → limita las predicciones al rango observado en el conjunto de entrenamiento, evitando valores fuera de escala.
- `"round_to_int": False` → permite redondear a enteros si el objetivo representa conteos (en este caso, se mantiene en valores continuos).

- `"lower"` y `"upper"` → almacenan los valores mínimo y máximo del objetivo (`y_train`) para el recorte (*clipping*).

La función `postprocess_preds()` aplica estas reglas de forma flexible:

1. Copia las predicciones (`yhat`).
2. Aplica recorte (`clip`) si corresponde.
3. Redondea si la política lo indica.
4. Devuelve las predicciones ajustadas (`ypp`).

Este paso asegura que las salidas del modelo sean **realistas y coherentes** con los datos de entrenamiento.

In [16]:
```
# ========================================
# 8) Política de decisión (mínima)
# ========================================
POLICY = {
    "clip_to_train_range": True,   # recorta predicciones al rango visto en TRAI
    "round_to_int": False,         # pon True si el objetivo es entero (conteos)
    "lower": float(y_train.min()),
    "upper": float(y_train.max()),
}
print("Política:", POLICY)

def postprocess_preds(yhat, policy=POLICY):
    ypp = yhat.copy()
    if policy.get("clip_to_train_range", False):
        ypp = np.clip(ypp, policy["lower"], policy["upper"])
    if policy.get("round_to_int", False):
        ypp = np.rint(ypp).astype(int)
    return ypp
```

```
Política: {'clip_to_train_range': True, 'round_to_int': False, 'lower': 12.0, 'up
per': 98.0}
```

Evaluación final del modelo seleccionado en conjunto TEST

Esta celda realiza la **evaluación final** del modelo ganador (seleccionado en la comparación anterior) utilizando el conjunto de **prueba (TEST)**, el cual no fue visto durante el entrenamiento ni la validación cruzada.

Proceso:

1. **Entrenamiento final:** se ajusta `winner_pipe` con todos los datos de entrenamiento (`X_train`, `y_train`).
2. **Predicción en TEST:** se generan las predicciones (`y_pred`) y se aplica la política de postprocesamiento (`postprocess_preds`).
3. **Cálculo de métricas de desempeño:**
   - **RMSE:** mide el error promedio ponderado (penaliza más los errores grandes).
   - **MAE:** error absoluto medio, más interpretable.

- **R^2:** coeficiente de determinación, que indica cuánto de la variabilidad del objetivo explica el modelo.

4. **Visualización rápida:** muestra una comparación entre los 10 primeros valores reales y predichos.

El resultado final resume el rendimiento real del modelo en datos completamente nuevos, validando si el ajuste previo se generaliza adecuadamente.

In [17]:
```
# ========================================
# 9) Evaluación final en TEST
# ========================================
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

winner_pipe.fit(X_train, y_train)
y_pred = winner_pipe.predict(X_test)
y_pp   = postprocess_preds(y_pred, POLICY)

rmse = mean_squared_error(y_test, y_pp, squared=False)
mae  = mean_absolute_error(y_test, y_pp)
r2   = r2_score(y_test, y_pp)

print(f"TEST → RMSE: {rmse:.4f} | MAE: {mae:.4f} | R²: {r2:.4f}")

# vistazo rápido (primeros 10)
preview = pd.DataFrame({
    "y_true": y_test.reset_index(drop=True),
    "y_pred": pd.Series(y_pp)
}).head(10)
print(preview.to_string(index=False))
```

```
TEST → RMSE: 6.0085 | MAE: 4.2134 | R²: 0.1696
 y_true    y_pred
     21 30.168871
     28 25.248185
     24 24.880594
     27 26.646117
     31 29.199765
     33 32.922259
     29 26.260012
     27 29.391451
     25 27.448959
     33 27.851471
```

Interpretabilidad del modelo y breve análisis de errores

En esta etapa final se busca **entender el comportamiento del modelo ganador** y analizar de forma breve los **errores de predicción**, con el objetivo de identificar posibles sesgos o patrones que afecten el rendimiento.

10.1 Evaluación de la política de postprocesamiento

Se revisa cuántas predicciones fueron **recortadas (clipped)** por la política definida previamente (`POLICY`):

- `clipped_low` : proporción de predicciones menores al rango de entrenamiento.
- `clipped_high` : proporción de predicciones mayores al rango de entrenamiento.

Esto permite verificar si el modelo tiende a generar valores fuera del rango observado.

10.2 Importancia de variables (Permutation Importance)

Se aplica la técnica de **Permutación de Importancia** (`permutation_importance`) para medir cuánto contribuye cada variable del conjunto original (`X_test`) a la calidad de las predicciones.

- Cada característica se altera aleatoriamente y se mide el impacto en el **RMSE**.
- Las variables que provocan un mayor deterioro del desempeño son las más relevantes.
- Se listan las **15 características más importantes**, junto con su desviación estándar, para entender qué atributos influyen más en el valor predicho del **IMC (BMI)**.

10.3 Análisis de errores

Se comparan los valores reales (`y_true`) con las predicciones (`y_pred`) y se calcula el **error absoluto** (`|y_true - y_pred|`).

- Se genera un resumen estadístico del error (percentiles 10, 25, 50, 75, 90).
- Se listan los **10 casos con mayor error**, junto con sus variables originales, para inspeccionar posibles patrones anómalos o puntos atípicos.

Este paso permite detectar si el modelo falla de forma sistemática en determinados rangos o perfiles.

10.4 Análisis por subgrupos (variable Diabetes_012)

Si el dataset incluye la columna **`Diabetes_012`** , se calcula el **Error Absoluto Medio (MAE)** por categoría:

- `0` : No diabético
- `1` : Prediabético
- `2` : Diabético

De esta forma se evalúa si el modelo mantiene un desempeño homogéneo entre grupos o si alguno presenta **mayor margen de error**, lo que podría reflejar un sesgo o necesidad de ajuste.

Este bloque de análisis aporta **transparencia e interpretabilidad**, mostrando no solo qué tan bien predice el modelo, sino también **cómo** y **dónde** comete errores.

In [18]:
```
# ========================================
# 10) Interpretabilidad + breve error analysis (mínimo)
# ========================================
import numpy as np
import pandas as pd
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_absolute_error

# 10.1 ¿Cuánto recorta la política?
raw_pred = winner_pipe.predict(X_test)
clip_low  = (raw_pred < POLICY["lower"]).mean()
clip_high = (raw_pred > POLICY["upper"]).mean()
print(f"[Policy] clipped_low: {clip_low:.3%} | clipped_high: {clip_high:.3%}")

# 10.2 Importancias por Permutación (sobre columnas ORIGINALES)
r = permutation_importance(
    winner_pipe,            # pipeline completa
    X_test, y_test,
    n_repeats=10,
    random_state=RANDOM_STATE,
    scoring="neg_root_mean_squared_error"
)

feat_names = X_test.columns  # mismos nombres que el X de entrada
imp = (pd.DataFrame({
        "feature": feat_names,
        "importance": r.importances_mean,
        "std": r.importances_std
     })
     .sort_values("importance", ascending=False)
     .head(15)
)
print("\nTop-15 importancias (perm, columnas originales):")
print(imp.to_string(index=False))

# 10.3 Errores: resumen + peores casos
y_hat = winner_pipe.predict(X_test)
y_pp  = postprocess_preds(y_hat, POLICY)
res   = pd.DataFrame({
    "y_true": y_test.reset_index(drop=True),
    "y_pred": pd.Series(y_pp)
})
res["abs_err"] = (res["y_true"] - res["y_pred"]).abs()
print("\nResumen de |error|:")
print(res["abs_err"].describe(percentiles=[.1,.25,.5,.75,.9]).to_string())

print("\nPeores 10 casos (|error| alto):")
top_bad_idx = res["abs_err"].nlargest(10).index
print(pd.concat([res.loc[top_bad_idx], X_test.reset_index(drop=True).loc[top_bad
      .to_string(index=False))

# 10.4 Métricas por subgrupos (adaptado a diabetes)
#     Usamos 'Diabetes_012' si existe; si no, se omite.
if "Diabetes_012" in X_test.columns:
    by_cls = (pd.concat([X_test.reset_index(drop=True)[["Diabetes_012"]], res],
              .groupby("Diabetes_012")["abs_err"]
              .agg(["count","mean","median"]))
    print("\nMAE por Diabetes_012:")
    print(by_cls.to_string())
```

```
[Policy] clipped_low: 0.000% | clipped_high: 0.000%

Top-15 importancias (perm, columnas originales):
          feature  importance      std
              Age    0.397274 0.007224
           HighBP    0.239223 0.007327
          GenHlth    0.203503 0.005077
         DiffWalk    0.163398 0.005420
              Sex    0.059393 0.001695
     PhysActivity    0.044706 0.001677
           Smoker    0.042054 0.002782
           Income    0.034854 0.001595
        Education    0.024730 0.001535
HvyAlcoholConsump    0.016819 0.001436
         HighChol    0.016190 0.000892
         PhysHlth    0.011707 0.001641
           Fruits    0.006543 0.001250
         MentHlth    0.006284 0.000762
           Stroke    0.004500 0.000664

Resumen de |error|:
count    50736.000000
mean         4.213426
std          4.283612
min          0.000024
10%          0.597673
25%          1.526833
50%          3.265953
75%          5.628112
90%          8.573774
max         72.172752

Peores 10 casos (|error| alto):
 y_true    y_pred   abs_err Diabetes_012  HighBP  HighChol  CholCheck  Smoker  St
roke  HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  HvyAlcoholConsump  Any
Healthcare  NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk    Sex  Age  Educa
tion  Income
     98 25.827248 72.172752          nan       0         1          1       1
0                    0            1       1        1                  0
1           0        2         0        20         0 Female 10.0          6
7
     95 24.390449 70.609551          nan       0         0          1       1
0                    0            1       1        1                  1
1           0        1         0         0         0 Female  3.0          6
6
     95 25.921093 69.078907          nan       0         1          1       0
0                    0            1       1        1                  0
1           0        2         0        30         0 Female  7.0          6
8
     92 24.857489 67.142511          nan       0         0          1       0
0                    0            1       1        1                  0
1           0        2         0         0         0 Female  2.0          6
8
     92 25.705187 66.294813          nan       0         0          1       0
0                    0            1       1        1                  0
1           0        2         3         0         0 Female  5.0          6
8
     92 26.346074 65.653926          nan       0         0          1       1
0                    0            1       1        1                  0
0           1        2        30         0         0 Female  2.0          4
```

```
7
     92 27.017581 64.982419          nan       1         1          1       0
0                    0             1       1        1                  0
1           0        1         0         0         0 Female 10.0          5
7
     92 27.466353 64.533647          nan       0         1          1       0
0                    0             1       1        1                  0
1           0        1         0         0         0 Female  6.0          4
3
     95 31.042341 63.957659          nan       0         1          1       1
0                    0             0       0        0                  0
0           1        2         0         6         1 Female  8.0          4
1
     87 23.567668 63.432332          nan       0         0          1       0
0                    0             1       1        1                  0
1           0        1         0         0         0 Female  4.0          6
8

MAE por Diabetes_012:
              count      mean    median
Diabetes_012
nan           50736  4.213426  3.265953
```

In [19]:
```
Xtr = winner_pipe.named_steps["prep"].transform(X_test)
model = winner_pipe.named_steps["model"]
r2 = permutation_importance(model, Xtr, y_test, n_repeats=10,
                            random_state=RANDOM_STATE,
                            scoring="neg_root_mean_squared_error")
feat_names_ohe = winner_pipe.named_steps["prep"].get_feature_names_out()
imp_ohe = pd.DataFrame({"feature": feat_names_ohe,
                        "importance": r2.importances_mean,
                        "std": r2.importances_std}).sort_values("importance", as
```

In [20]:
```
imp_ohe
```

Out[20]:

	feature	importance	std
16	num__Age	0.371321	0.007316
0	num__HighBP	0.240095	0.007120
12	num__GenHlth	0.195548	0.005575
15	num__DiffWalk	0.155654	0.005170
3	num__Smoker	0.041787	0.002687
6	num__PhysActivity	0.040833	0.001321
17	num__Education	0.021622	0.001534
9	num__HvyAlcoholConsump	0.015396	0.001323
1	num__HighChol	0.014924	0.000695
18	num__Income	0.009980	0.001331
14	num__PhysHlth	0.009497	0.001729
7	num__Fruits	0.007376	0.001351
4	num__Stroke	0.004338	0.000656
13	num__MentHlth	0.003907	0.000602
2	num__CholCheck	0.003620	0.000571
5	num__HeartDiseaseorAttack	0.001907	0.000740
11	num__NoDocbcCost	0.001571	0.000544
8	num__Veggies	0.000627	0.000914
10	num__AnyHealthcare	0.000329	0.000351
19	cat__Diabetes_012_nan	0.000000	0.000000

In [ ]: